AD-A099 358 WISCONSIN UNIV-MADISON MATHEMATICS RESEARCH CENTER F/G 9/2
THE SOLUTION OF LINEAR COMPLEMENTARITY PROBLEMS ON AN ARRAY PRO--ETC(U)
JAN 81 C W CRYER, P M FLANDERS, D J HUNT DAAG29-80-C-0041
UNCLASSIFIED MRC-TSR-2170 NL

END
DATE FILMED
6-81
DTIC

(9) MRC Technical Summary Rep[illegible].70

(1) THE SOLUTION OF LINEAR COMPLEMENTARITY PROBLEMS ON AN ARRAY PROCESSOR,

(10) C. W. Cryer / P. M. Flanders
D. J. Hunt / S. F. Reddaway
and J. Stansbury

(11) Jan 81

(12) 61

(14) MRC-TSR-2170

(15) DAAG29-80-C-0041
NSF-MCS77-26732

Mathematics Research Center
University of Wisconsin—Madison
610 Walnut Street
Madison, Wisconsin 53706

January 1981

(Received September 25, 1980)

Approved for public release
Distribution unlimited

Sponsored by

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, D. C. 20550

UNIVERSITY OF WISCONSIN-MADISON
MATHEMATICS RESEARCH CENTER

THE SOLUTION OF LINEAR COMPLEMENTARITY PROBLEMS ON AN ARRAY PROCESSOR

C. W. Cryer[*,1,2], P. M. Flanders[+], D. J. Hunt[+], S. F. Reddaway[+], and J. Stansbury[**,1]

Technical Summary Report #2170
January 1981

ABSTRACT

The Distributed Array Processor (DAP) manufactured by International Computers Limited is an array of 1-bit 200-nanosecond processors. The Pilot DAP on which the present work was done is a 32 × 32 array; the commercially available machine is a 64 × 64 array. We show how the projected SOR algorithm for the linear complementarity problem $Aw \geq b$, $w \geq 0$, $w^T(Aw - b) = 0$, can be adapted for use on the DAP when A is the 'finite-difference' matrix corresponding to the difference approximation to the Laplace operator. Application is made to two linear complementarity problems arising, respectively, from two-and three-dimensional porous flow free boundary problems.

AMS (MOS) Subject Classifications: 35J25, 35J65, 65K05, 65M99, 68A10

Key Words: Array processor, Parallel computation, Free boundary problem, Variational inequality, Porous flow, Linear complementarity problem, Numerical solution, Successive over-relaxation

Work Unit Number 3 (Numerical Analysis and Computer Science)

---

[*]Computer Sciences Department and Mathematics Research Center, University of Wisconsin-Madison, Madison, Wisconsin, U.S.A.

[+]Research and Advanced Development Center, International Computers Limited, Fairview Road, Stevenage, Hertfordshire SG1 2DX, England.

[**]Computer Sciences Department, University of Wisconsin-Madison, Madison, Wisconsin, U.S.A.

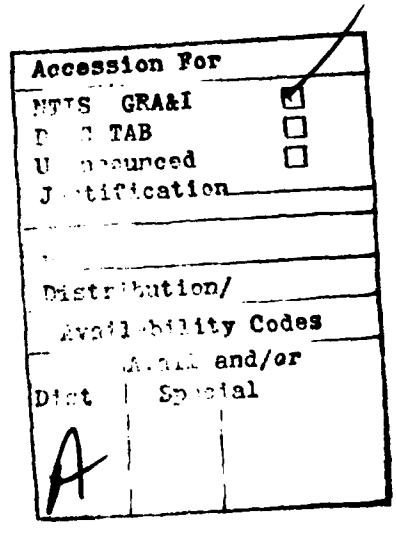
Sponsored by:

[1]The National Science Foundation under Grant No. MCS77-26732;
[2]The United States Army under Contract No. DAAG29-80-C-0041.

SIGNIFICANCE AND EXPLANATION

An array processor, the Distributed Array Processor manufactured by International Computers Limited, has recently become available. The DAP is an array of 1-bit processors; in the production machine there are 4096 processors arranged as a 64 × 64 array. It is normally programmed in an array processing extension of Fortran.

It is of interest to develop algorithms which can efficiently use the great computing capacity of the DAP. We show how the projected SOR algorithm for the linear complementarity problem $Aw \geq b$, $w \geq 0$, $w^T(Aw - b) = 0$ can be adapted for use on the DAP when A is the finite-difference matrix corresponding to the difference approximation to the Laplace operator. Application is made to two linear complementarity problems arising, respectively, from two and three-dimensional porous flow free boundary problems.



THE SOLUTION OF LINEAR COMPLEMENTARITY PROBLEMS ON AN ARRAY PROCESSOR

C. W. Cryer[*,1,2], P. M. Flanders[+], D. J. Hunt[+], S. F. Reddaway[+], and J. Stansbury[**,1]

1. Introduction

An LCP (linear complementarity problem) is a problem of the form: Find a real n-vector $w = (w_i)$ satisfying

$$\begin{aligned} &\text{(a)} && Aw \geq b , \\ &\text{(b)} && w \geq 0 , \\ &\text{(c)} && w^T(Aw - b) = 0 , \end{aligned} \tag{1.1}$$

where $b = (b_i)$ is a known real n-vector and $A = (a_{ij})$ is a known real $n \times n$ matrix.

Linear complementarity problems arise in many contexts (Balinski and Cottle [1978]). In particular, there is a close connection between linear complementarity problems and variational inequalities (Cottle, Giannessi and Lions [1980], Cryer and Dempster [1980]).

Many problems in continuum mechanics can be reformulated as variational inequalities (Duvaut and Lions [1976], Kinderlehrer and Stampacchia [1980]), which, when discretized, reduce to linear complementarity problems of the form (1.1) with special features:

---

[*]Computer Sciences Department and Mathematics Research Center, University of Wisconsin-Madison, Madison, Wisconsin, U.S.A.

[+]Research and Advanced Development Center, International Computers Limited, Fairview Road, Stevenage, Hertfordshire SG1 2DX, England.

[**]Computer Sciences Department, University of Wisconsin-Madison, Madison, Wisconsin, U.S.A.

Sponsored by:

[1]The National Science Foundation under Grant No. MCS77-26732;
[2]The United States Army under Contract No. DAAG29-80-C-0041.

1. A is a large matrix, perhaps of order 25,000.

2. A is a 'finite-difference' or 'finite-element' matrix; in particular,

   A is sparse with a great deal of structure. (1.2)

3. A large percentage of the elements of the solution w are non-zero.

Because of these special features, the standard methods of solving linear complementarity problems are not very efficient, and methods of solution have been developed which take advantage of the structure of A: projected SOR (Cryer [1971], Glowinski [1971]); modified block SOR (Cottle, Golub, and Sacher [1978]); multigrid projection (Brandt and Cryer [1980]); and generalizations of projected SOR (Mangasarian [1977]). Cryer [1980] briefly surveys much of this work.

In the present paper we consider the use of the parallel computer DAP to solve linear complementarity problems with the features (1.2). The DAP (Distributed Array Processor, manufactured by International Computers Limited), which is an SIMD array of typically 64 × 64 processors, is described in Section 2. In Section 3 we describe the implementation on the DAP of projected SOR to solve a linear complementarity problem derived from a two-dimensional porous flow free boundary problem, and in Section 4 we extend this work and solve a linear complementarity problem derived from a three-dimensional porous flow free boundary problem. In Section 5 we comment on possible future developments, and the overall conclusions are in Section 6.

2. The Pilot DAP (Distributed Array Processor)

The present work was carried out on the Pilot 32 × 32 DAP at Stevenage, England, and we will describe this machine first. A 64 × 64 version is available, and the minor differences between the two machines are indicated at the end of this section.

DAP Hardware

The essential features of the Pilot DAP hardware are as follows (Flanders, Hunt, Reddaway, and Parkinson [1977], Reddaway [1979]):

1. A 32 × 32 array of identical processing elements (PEs) with a cycle time of 200 nanoseconds.
2. Each PE has a one-bit adder, 2K bits of storage, and three one-bit registers (a general purpose register for accessing data and performing arithmetic; a carry register; and an activity control register).
3. Each PE is connected to its four neighboring PE's (North, South, East, and West). In a given cycle all PE's access their neighbor in the same direction (determined by the program). In addition, the PEs are linked by row and column highways which connect together all the PEs in each row and column.
4. There is a master control unit (MCU) which broadcasts instructions to all the PEs. All PEs can perform the same instruction simultaneously, but certain instructions are only effective if the activity control register is 'true'.

DAP Software

A programme to run on a DAP system normally comprises a standard FORTRAN program and a number of subroutines and functions written in an array processing extension at FORTRAN known as DAP-FORTRAN (Flanders [1979], Gostick [1979], ICL [1979]).

The standard FORTRAN is executed by the host computer and provides mainly input-output and overall control. The DAP-FORTRAN is executed by the DAP and provides high speed computation. Data is shared between them using common blocks held in DAP store.

Some features of DAP-FORTRAN are described below.

In addition to the data types of FORTRAN, DAP-FORTRAN has two new data types: vector and matrix. With a 32 × 32 DAP a vector has 32 components and a matrix has 32 × 32 components; the components can be real, integer, or logical.

For example, the data statements

```
REAL  U( ), V( , ), W(,5), X(,,3),
INTEGER  A(,1), B( ), C(,,4)                    (2.1)
LOGICAL FLAGS(,2), MASK( , )
```

declare U (a real vector), V (a real matrix), W (an array of 5 real vectors), X (an array of 3 real matrices), A (an array of 1 integer vector), B (an integer vector), C (an array of 4 integer matrices), FLAGS (an array of 2 logical vectors), and MASK (a logical matrix).

Expressions in DAP-FORTRAN can consist of scalars, vectors, and matrices with the usual unary and binary operations. Operations on vectors and matrices are performed in parallel using all 32 × 32 PEs.

Operations between a scalar and a vector or a matrix cause implicit expansion of the scalar to the necessary dimensions. For example, if M is a matrix of size 32 × 32 and S is a scalar, then M = M + S causes S to be implicitly expanded to size 32 × 32 with each element being equal to S; then the corresponding elements of "matrix" S and matrix M are added in parallel and assigned to M in parallel.

Arrays of vectors and matrices may be used to construct more complex structures. To process a vector or matrix array requires performing calculations on the individual vectors or matrices in the array.

Selection and updating of parts of vectors and matrices can be performed using the powerful indexing capabilities of DAP-FORTRAN. Matrix sections can be specified by omitting subscripts along which all elements are to be taken. Using this, whole rows or columns can be selected from matrices. For example, M(I,) specifies the I-th row of matrix M.

Shift indexing is a very useful feature of DAP-FORTRAN. For example, in a simple solution of Laplace's equation on a 32 × 32 grid we wish to replace each element with the average of its four neighbors. This could be coded in FORTRAN as:

```
      DO  10  I = 2,31
      DO  10  J = 2,31
       Y(I,J) = (X(I + 1,J) + X(I - 1,J) + X(I,J + 1) + X(I,J - 1)) / 4.0
      10  continue
```

Further code would be needed to handle elements on the edges of the matrix.

The DAP-FORTRAN code is much simpler:

```
X = (X(+,) + X(-,) + X(,+) + X(,-)) / 4.0 .          (2.1)
```

The term X(+,) uses shift indexing. In particular, X(+,) specifies a matrix where the (I,J) element is the (I + 1,J) element of X, for $1 \leq I \leq 32$ and $1 \leq J \leq 32$. Thus, X(+,) contains all the "south" neighbors of X. Edge values (corresponding to subscripts 0 or 33) are defined to be zero. As an alternative, cyclic geometry may be specified by using a GEOMETRY statement.

Longer shifts can be performed by explicit system functions; for example, SHS(X,I) shifts the matrix X I positions to the south. Note that since all the updating is performed simultaneously, it is not necessary to write the results to another matrix.

Logical matrices and vectors can be used to select elements from an array. For example, if we wished to update only certain elements of X in statement (2.1), we could set the corresponding elements of LM, a logical matrix, to true and all other elements of LM to false. That is, if X(I,J) is to contain the average of its four neighbors, then LM(I,J) is set to true. Otherwise, LM(I,J) is false. Then the following statement performs the required task:

```
X(LM) = (X(+,) + X(-,) + X(,+) + X(,-)) / 4.0 .
```

DAP-FORTRAN has a number of useful system functions whose arguments and results may be scalars, vectors, or matrices. The ALTC, ALTR, MERGE, MAX, and ABS functions will be briefly described since these are used in the programs in this paper.

The functions ALTC and ALTR return logical matrices. If C is the argument to ALTC, then the first C columns of the result matrix are set to false, the next C columns to true, the next C columns to false, etc. ALTR performs similarly for rows.

The function MAX (now named MAXV) returns a scalar equal to the largest number in its vector or matrix argument. The function ABS returns a vector or matrix containing the absolute value of every element in its argument.

The function MERGE takes three arguments and returns a matrix. The first two arguments are matrices (or implicitly expanded scalars) and the third argument is a logical matrix. If the (I,J) element of the logical matrix is true then the (I,J) element of the result matrix is set equal to the (I,J) element of the first matrix; otherwise, it is set equal to the (I,J) element of the second matrix.

Examples of DAP-FORTRAN programs are given in Sections 3 and 4, and the Appendices.

DAP Arithmetic

When a DAP-FORTRAN program is executed by the DAP, expressions involving only scalars are executed sequentially, but operations on vectors and matrices are performed in parallel by the PEs.

The DAP memory can be visualized as a cuboid, with 2K horizontal planes, each plane being a 32 × 32 square of bits. The 32 × 32 array of PEs lies on top of the cube, and each column of 2K bits belongs to the PE above it.

Two storage modes are used in DAP-FORTRAN: vertical and horizontal. Scalars and vectors are stored in horizontal mode while matrices are held in vertical mode.

In vertical mode, each number is held entirely within the store of one PE with successive bits in successive store locations. Thus, for an integer matrix, the

sign bit of every element in the matrix would be held in the same store address of each PE.

In horizontal mode, a number is spread along a row of PEs. Thus, a scalar occupies one row while a vector occupies 32 rows. DAP instructions are also stored in this format.

All arithmetic is carried out using subroutines. Some operation times for 32 bit numbers are given in Table 2.1.

It will be noted that vector arithmetic is faster than matrix arithmetic. This is because a row of PEs are available for each vector component, while only one PE is available for each matrix component.

Some of the quoted computation times are data dependent. In particular, matrix multiplication by a scalar typically varies from 170μs to 200μs depending upon the distribution of zeros in the binary representation of the constant; for special scalars such as .5 or 3 the multiplication time can be as low as 60μs.

Operation	Matrix	Vector	Scalar
floating point addition	140-180μs	54μs	27μs
floating point multiplication	315μs	50μs	34μs
floating point multiplication by a scalar	60-200μs	40μs	-
One shift of a real matrix, e.g. X(+,).	15μs	2μs	-
Move a floating point matrix	15μs	2μs	2μs
logical AND	2μs	2μs	2μs
logical mask	1μs	2μs	-

Note: Times are slightly different on production DAPs.

Table 2.1. Average DAP-FORTRAN arithmetic times for the Pilot DAP.

Host-DAP Interface

The sequence of operations for compiling and running DAP programs is as follows.

(a) The host computer compiles the host FORTRAN program and the DAP-FORTRAN subroutines into host and DAP machine codes respectively.

(b) DAP machine code, incorporating all necessary low level subroutines, is loaded into DAP memory in horizontal mode where it occupies a few bits of each PE's memory. Host machine code is loaded into the host memory.

(c) Execution begins in the host and control is transferred to the DAP as required by subroutine calls. On completion of DAP processing the host resumes execution at the point following the call.

Detailed information on the Pilot DAP relevant to understanding the programs in this paper is given in Appendix A.

The Production DAP

The current production DAP is generally similar to the Pilot but differs as follows:

(a) there are 4096 PEs arranged in a 64 × 64 array;

(b) each PE has 4K bits of memory;

(c) arithmetic operations differ somewhat in timing but are overall a little faster;

(d) coupling between host and DAP is more direct so the interface is simpler than indicated in Appendix A.

3. Numerical solution of a two-dimensional free boundary problem

The flow of water through a porous dam is a well-known model problem. Water seeps from a reservoir of height H through a rectangular dam of width L to a reservoir of height h. Part of the dam is saturated and the remainder of the dam is dry. The wet and dry regions are separated by an unknown free boundary Γ which must be found as part of the solution.

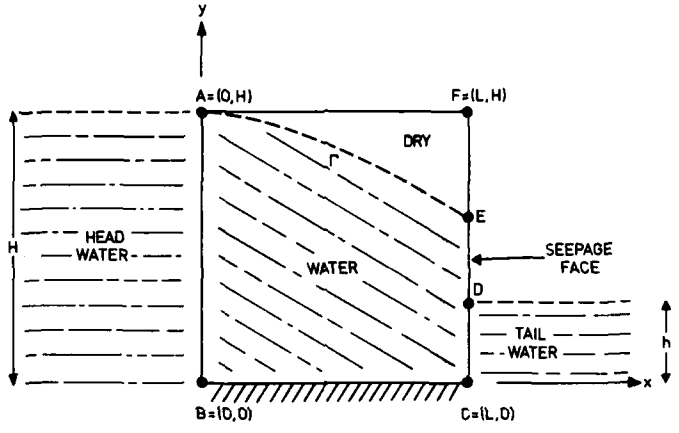


FIG. 3.1. Flow through a porous rectangular dam R.

As shown by Baiocchi [1972] the problem can be formulated as follows: Find u on the rectangle $R = ABCF$ such that

$$\begin{aligned} &\text{(a)} && -\nabla^2 u \geq -1, \quad \text{on} \quad R, \\ &\text{(b)} && u \geq 0, \quad \text{on} \quad R, \\ &\text{(c)} && u(-\nabla^2 u + 1) = 0, \quad \text{on} \quad R, \end{aligned} \tag{3.1}$$

and

$$u = g = \begin{cases} (H - y)^2/2 & \text{on} \quad AB, \\ (h - y)^2/2 & \text{on} \quad CD, \\ [H^2(L - x) + h^2 x]/2L, & \text{on} \quad BC, \\ 0, & \text{on} \quad DFA. \end{cases} \tag{3.2}$$

The wet region of the dam consists of the points where $u > 0$ and the dry region consists of the points where $u = 0$.

When the problem (3.1), (3.2) is approximated using the classical five point difference approximation for the Laplace operator, one obtains an LCP of the form (1.1), where the matrix A and right hand side b are the same as those that would be obtained if the Dirichlet problem

$$\begin{aligned} -\nabla^2 u &= -1, \quad \text{on} \quad R, \\ u &= g, \quad \text{on} \quad R, \end{aligned} \tag{3.3}$$

were approximated by the finite difference equation $Aw = b$. More precisely, let an $M \times N$ grid with gridlength Δx be superimposed upon R, and let the values of u and g at the point $([j-1]\Delta x, [i-1]\Delta x)$ be denoted by u_{ij} and g_{ij}, respectively, for $1 \le i \le M$ and $1 \le j \le N$. Then (1.1) takes the form

$$\text{(a)} \quad 4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} \ge -(\Delta x)^2 \quad \text{for } 1 < i < M,\ 1 < j < N,$$

$$\text{(b)} \quad w_{ij} \ge 0, \quad \text{for } 1 < i < M,\ 1 < j < N, \tag{3.4}$$

$$\text{(c)} \quad w_{ij}(4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} + (\Delta x)^2) = 0, \quad \text{for } 1 < i < M,\ 1 < j < N,$$

$$\text{(d)} \quad w_{ij} = g_{ij}, \quad \text{for } ((j-1)\Delta x, (i-1)\Delta x) \in \partial R.$$

We discuss below two iterative methods for solving (3.4): the projected Jacobi method and the projected SOR method. The projected Jacobi method is much slower than the projected SOR method, but is trivial to implement on the DAP and serves as a useful introduction to DAP programming.

The projected Jacobi method

Let $w^{(0)} = (w_{ij}^{(0)})$ be an initial guess for the solution $w = (w_{ij})$ of (3.4). One generates a sequence of approximations $w^{(k)} = (w_{ij}^{(k)})$, $k = 1,2,\ldots,$

(a) $$z_{ij}^{(k)} = w_{i-1,j}^{(k)} + w_{i+1,j}^{(k)} + w_{i,j-1}^{(k)} + w_{i,j+1}^{(k)} - (\Delta x)^2 ,$$

(b) $$w_{ij}^{(k+\frac{1}{2})} = z_{ij}^{(k)}/4 ,$$

(c) $$w_{ij}^{(k+1)} = \max(0, w_{ij}^{(k+\frac{1}{2})}) , \qquad (3.5)$$

$$\text{for } 1 < i < M \text{ and } 1 < j < N ;$$

(d) $$w_{ij}^{(k+1)} = g_{ij}, \quad \text{for } ((j-1)\Delta x, (i-1)\Delta x) \in \partial R .$$

It is known that the projected Jacobi method will converge (Mangasarian [1977]).

If $M \leq 32$ and $N \leq 32$ then the gridpoints can be regarded as a subset of a 32×32 array, and one PE can be associated with each gridpoint. Defining $w^{(k)}$, $w^{(k+1)}$ and $z^{(k)}$ as real DAP-FORTRAN matrices, the computation (3.5) is trivial to implement on the DAP.

In Figure 3.2 we list a DAP subroutine JACOBI which solves the dam problem for the case $h = 0$, $H = 31$, $L = 31$, $M = N = 32$, and $\Delta x = 1$. This subroutine could be called by a host program, which could then print the answers in the matrix W.

Using the operation times given in Table 2.1 we can readily estimate the time required per iteration in the main loop of the JACOBI subroutine (see Figure 3.3).

```
      SUBROUTINE      JACOBI

      LOGICAL MASK( , ), WSIGN( , )     Declare logical 32 × 32 matrices, MASK and
                                        WSIGN
      REAL  W( , ) , Z( , )             Declare real floating point 32 × 32 matrices
                                        W  and  Z

      REAL    INDEX( )                  Declare a real floating point 32-vector INDEX.
      EQUIVALENCE (W,WSIGN)             Declare the logical matrix WSIGN equivalent to
                                        the first bit, the sign bit, of the matrix  W.
      HEIGHT = 31.0

      WIDTH = 31.0

      DO  10    I = 1,32                Initialize INDEX vector.

      INDEX(I) = (32 - I)/31.0

10    CONTINUE

      W = 0                             Clear matrix  W

      TEMP = HEIGHT*HEIGHT*.5

      W(1, ) = TEMP * INDEX             Set values of the matrix  W  equal to  g
                                        on bottom  (BC).

      W( ,1) = TEMP * INDEX * INDEX     Set values of the matrix  W  equal to  g
                                        on left  (AB).
      MASK = .TRUE.

      MASK(1, ) = .FALSE.
                                        Set the matrix MASK to be true at interior
      MASK(32, ) = .FALSE.              points and false at boundary points.

      MASK( ,1) = .FALSE.

      MASK( ,32) = .FALSE.

      DO  50  I = 1, 100                Start of main loop

 1    Z = W(+, ) + W(-,) + W( ,+)       Sum neighbors and store in  Z  matrix.
                 + W( ,-) - 1.0

 2    W(MASK) = .25*Z                   Transfer average to  W  at interior points.

 3    W(MASK .AND. WSIGN) = 0.0         Project by setting  W = 0  at points where
                                        MASK is true and the sign of  W  is negative.
50    CONTINUE

      END
```

FIG. 3.2. The DAP subroutine JACOBI.

Statement	Operations	Time (μs)
Z = W(+,) + W(-,) + W(,+) + W(,-) - 1.0	4 floating point matrix additions/subtractions	640
	4 index shifts	60
	1 scalar-matrix assignment	15
W(MASK) = .25*Z	1 floating point matrix multiplication by a special constant	70
	1 logical mask	1
W(MASK .AND. WSIGN) = 0.0	1 logical AND	2
	1 logical mask	1
	1 scalar-matrix assignment	15
DO 50 I = 1,100		7
		811

FIG. 3.3. Estimated computation time for the main loop of JACOBI.

From Figure 3.3 we see that one projected Jacobi iteration over the whole 32×32 grid requires 811μs.

The projected SOR method

Let $w^{(0)} = (w_{ij}^{(0)})$ be an initial guess for the solution $w = (w_{ij})$ of (3.4). In the usual implementation of projected SOR one generates a sequence of approximations $w^{(k)} = w_{ij}^{(k)}$ as follows:

(a) $$z_{ij}^{(k)} = w_{i-1,j}^{(k+1)} + w_{i+1,j}^{(k)} + w_{i,j-1}^{(k+1)} + w_{i,j+1}^{(k)} - (\Delta x)^2 ,$$

(b) $$w_{ij}^{(k+1/2)} = w_{ij}^{(k)} + \omega(z_{ij}^{(k)} - 4w_{ij}^{(k)}) / 4 ,$$

$$= (\omega/4)z_{ij}^{(k)} + (1 - \omega)w_{ij}^{(k)} , \qquad (3.6)$$

(c) $$w_{ij}^{(k+1)} = \max\{0, w_{ij}^{(k+1/2)}\},$$

$$\text{for } 1 < i < M \text{ and } 1 < j < N ,$$

where ω is a constant, the over-relaxation parameter.

It is known that the iteration (3.6) converges for all initial guesses $w^{(0)}$ iff $0 < \omega < 2$ (Cryer [1971], Glowinski [1971]).

The implementation (3.6) is not suitable for parallel computation because the new values $w^{(k+1)}$ cannot be computed simultaneously: $w_{i-1,j}^{(k+1)}$ and $w_{i,j-1}^{(k+1)}$ must be known before $w_{ij}^{(k+1)}$ can be computed.

However, there is a simple but ingenious way of making SOR suitable for parallel computation. In the implementation (3.6), we order the gridpoints by rows and columns (Figure 3.4a). Instead, let us visualize the gridpoints as forming a red-black chess board and number first the red points and then the black points (Figure 3.4b).

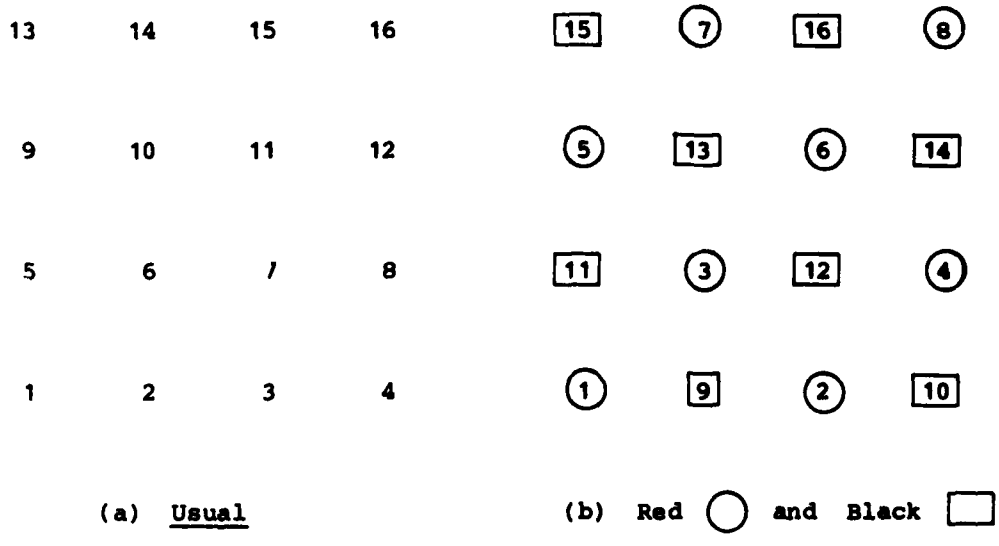


FIG. 3.4. Orderings of gridpoints (for a 4 × 4 grid).

Applying projected SOR to the points numbered as in Figure 3.4(b) we find that each projected SOR iteration can be broken down into two stages: in the red(first) stage projected SOR is applied to the red points; and in the black(second) stage projected SOR is applied to the black points:

Red Stage

$$\text{(a)} \qquad z_{ij}^{(k,\text{red})} = w_{i+1,j}^{(k,\text{black})} + w_{i-1,j}^{(k,\text{black})} + w_{i,j+1}^{(k,\text{black})} + w_{i,j-1}^{(k,\text{black})} - (\Delta x)^2 ,$$

$$\text{(b)} \qquad w_{i,j}^{(k+\frac{1}{2},\text{red})} = (\omega/4) z_{ij}^{(k,\text{red})} + (1-\omega) w_{ij}^{(k,\text{red})} , \tag{3.7}$$

$$\text{(c)} \qquad w_{i,j}^{(k+1,\text{red})} = \max\{0, w_{i,j}^{(k+\frac{1}{2},\text{red})}\} .$$

Black stage

$$\text{(a)} \qquad z_{ij}^{(k,\text{black})} = w_{i+1,j}^{(k+1,\text{red})} + w_{i-1,j}^{(k+1,\text{red})} + w_{i,j+1}^{(k+1,\text{red})} + w_{i,j-1}^{(k+1,\text{red})} - (\Delta x)^2 ,$$

$$\text{(b)} \qquad w_{ij}^{(k+\frac{1}{2},\text{black})} = (\omega/4) z_{ij}^{(k,\text{black})} + (1-\omega) w_{ij}^{(k,\text{black})} , \tag{3.8}$$

$$\text{(c)} \qquad w_{ij}^{(k+1,\text{black})} = \max\{0, w_{ij}^{(k+\frac{1}{2},\text{black})}\} .$$

Each stage can be carried out in parallel, with the red(black) processors working and the black(red) processors idle.

This idea of using the red-black ordering for parallel processors has appeared several times in the literature (Heller [1978]). Its use on DAP was first suggested by Hunt [1974]. (In Europe, white-black chessboards are more usual than red-black ones).

In Figure 3.5 we list a DAP-FORTRAN subroutine PROJSOR for implementing the heart of the algorithm (3.7), (3.8). The subroutine is provided with several input parameters with obvious meanings. In addition, two logical matrices are provided as input: the logical matrix MASKMASK is true at gridpoints in the interior of the dam, and false elsewhere; the logical matrix MASK is true at black gridpoints and false at red gridpoints. Finally, the values of the real matrix W at the boundary points ∂R must be computed using (3.4d) before PROJSOR is called. A full listing of the program is given in Appendix B.

```
      SUBROUTINE      PROJSOR

      COMMON /RMAT/W( , )
      COMMON /RSCA/MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
      COMMON /ISCA/NUMB ITERATIONS,NUMB ROWS,NUMB COLS
      COMMON /SUBLMAT/MASK( , ), MASK MASK( , )
      REAL W,MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
      LOGICAL MASK, MASK MASK
      INTEGER NUMB ITERATIONS,NUMB ROWS,NUMB COLS

      REAL Z( , ), GRID2, ZMIN W( , ),SAVEW( , )   Local variables.
      REAL ALPHA,BETA
      INTEGER NUMB TIMES
      LOGICAL DONE, WSIGN( , )
      EQUIVALENCE (WSIGN, W)

      W(WSIGN) = 0.0                               Ensure that  W  is nonnegative
                                                   everywhere.

      ALPHA = OMEGA * .25                          Calculate the constants that are
      BETA = 1.0 - OMEGA                           needed later on.
      GRID2 = (DAM HEIGHT/NUMB ROWS) ** 2

40    SAVE W = W                                   Start main loop.
      NUMB ITERATIONS = NUMB ITERATIONS + 1        Save the old value of  W.
      DO  45  NUMB TIMES = 1,2

 1    MASK(MASK MASK) = .NOT. MASK                 Reverse state of MASK.

 2    Z = W + W(-,-)                               Calculate  Z  on only the red (or
 3    Z = Z(+, ) + Z( ,+) - GRID2                  black) points as determined by
 4    W(MASK) = ALPHA * Z + BETA * W               the MASK.

 5    W(WSIGN .AND. MASK MASK) = 0.0               Project
45    CONTINUE

      MAX DIFF = MAX(ABS(SAVEW - W))               Find maximum difference between
                                                   old and new.

      DONE = (MAX DIFF.LE.EPSILON)                 Check if desired accuracy is
      IF (.NOT. DONE) GO TO 40                     attained.

      RETURN
```

FIG. 3.5. The DAP subroutine PROJSOR.

The computation time for one pass through the main loop of the subroutine PROJSOR is estimated in Figure 3.6, from which it follows that each PROJSOR iteration, which requires two passes through the loop, takes about $2 \times 1135 = 2.27$ms. To check this estimate, the average execution time per iteration in the subroutine PROJSOR was obtained by measuring (on a real external physical clock) the time required for a large number of iterations for the dam problem with $H = 24$, $h = 0$, $L = 16$, and $\Delta x = 1$. (This particular problem was chosen because it is a test problem which has been solved by many authors). The measured time per iteration on the pilot DAP was 2.2ms, as compared to the estimated time of 2.27ms.

Statement	Operations	Time (μs)
1 MASK(MASKMASK) = .NOT. MASK	1 logical mask	1
	1 logical negation	1
	1 logical store	1
2 Z = W + W(-,-)	1 index shift of two places	21
	1 floating point matrix addition	160
3 Z = Z(+,) + Z(,+) - GRID2	2 index shifts	30
	1 floating point matrix addition	160
	1 floating point matrix subtraction	160
	1 scalar-matrix assignment	15
4 W(MASK) = ALPHA * Z + BETA * W	1 floating point matrix addition	160
	2 floating point matrix multiplications by a constant	400
	1 logical mask	1
5 W(WSIGN .AND. MASK MASK) = 0.0	1 logical AND	2
	1 logical mask	1
	1 scalar-matrix assignment	15
DO 15 NUMB TIMES = 1,2		7
		1135

FIG. 3.6. Estimated computation time for the inner loop of PROJSOR.

We conclude this section with some comments:

1. For comparison, the dam problem with H = 24, h = 0, L = 16, and $\Delta x = 1$ was also solved on the UNIVAC 1180 at the University of Wisconsin, using the conventional ordering of gridpoints and an optimizing compiler with single precision arithmetic (36 bits), and the time per iteration was found to be 5.29ms. For this problem the Pilot DAP was therefore 2.4 times faster than the UNIVAC 1180.

   It should be noted that for this problem only $25 \times 17 = 425$ of the 1024 DAP PEs were used. On a square region the Pilot DAP would be six times faster than the UNIVAC 1180.

2. In general, one expects to be able to predict DAP execution times to within about 5%, because DAP programs have little overhead and spend almost all their time in computation.

3. Since DAP floating point operations are relatively expensive, it is worthwhile optimizing the code. (Readers who used early computers which also had relatively slow arithmetic operations may feel nostalgic). An example of such optimization occurs in the subroutine PROJSOR (see Figure 3.5). The computation (3.7a) could have been implemented as:

$$Z = W(+,\ ) + W(-,\ ) + W(\ ,+) + W(\ ,-) - GRID2$$

which requires three additions and one subtraction, and takes

$$\underset{\text{(shifts)}}{4(15)} + \underset{\text{(additions)}}{4(160)} + \underset{\text{(scalar-matrix assignment)}}{15} = 745\mu s\ .$$

However, by sharing intermediate results between PE's, the amount of arithmetic can be reduced; the implementation in PROJSOR is

$$Z = W + W(-,-)$$

$$Z = Z(+,\ ) + Z(+,\ ) - GRID2$$

which is estimated at only 546μs.

It should be noted that both implementations use only half the PEs for arithmetic at any one time. Larger grids or three-dimensional problems (see Section 4) can use all the PEs simultaneously.

4. Numerical solution of a three-dimensional free boundary problem.

A three-dimensional extension of the dam problem of Figure 3.1 was introduced by Stampacchia [1974] (see also France [1974]). Water seeps through a porous dam in a rectangular channel of width a and height H. The walls of the dam are vertical but the thickness of the dam is variable, so that the dam occupies the region

$$\Omega_3 = \Omega_2 \times (0,H) , \tag{4.1}$$

where the horizontal cross-section Ω_2 is of the form

$$\Omega_2 = \{(x,y) : 0 < x < a, \quad \varphi_1(x) < y < \varphi_2(x)\} . \tag{4.2}$$

In the specific problem considered here, Ω_2 is the L-shaped region

$$\Omega_2 = (0,ED) \times (0,FE) \cup [ED,AF) \times (0,AB) , \tag{4.3}$$

where the points A, B, C, D, E and F are as shown in Figure 4.1. The upstream water height is H and the downstream water height is h.

As shown by Stampacchia [1974], the problem can be formulated as follows:

Find u on the region Ω_3 such that:

$$\text{(a)} \qquad -\nabla^2 u = -[u_{xx} + u_{yy} + u_{zz}] \geq -1, \quad \text{in } \Omega_3 ,$$

$$\text{(b)} \qquad u \geq 0, \quad \text{in } \Omega_3 , \tag{4.4}$$

$$\text{(c)} \qquad u(-\nabla^2 u + 1) = 0, \quad \text{in } \Omega_3 ,$$

and

$$u = g = \begin{cases} \frac{1}{2}(H - z)^2, & \text{on the upstream face } AA_0F_0F , \\ \frac{1}{2}(h - z)^2, & \text{on the downstream face below water level } B_0C_0D_0E_0E_1D_1C_1B_1 , \\ 0 , & \text{on the downstream face above water level } B_1C_1D_1E_1EDCB , \\ 0 , & \text{on the top } ABCDEF , \\ \alpha(x,y), & \text{on the bottom } A_0B_0C_0D_0E_0F_0 \end{cases} \tag{4.5}$$

and

$$u_x \equiv u_n = 0, \quad \text{on the sides } ABB_0A_0 \text{ and } EFF_0E_0 . \tag{4.6}$$

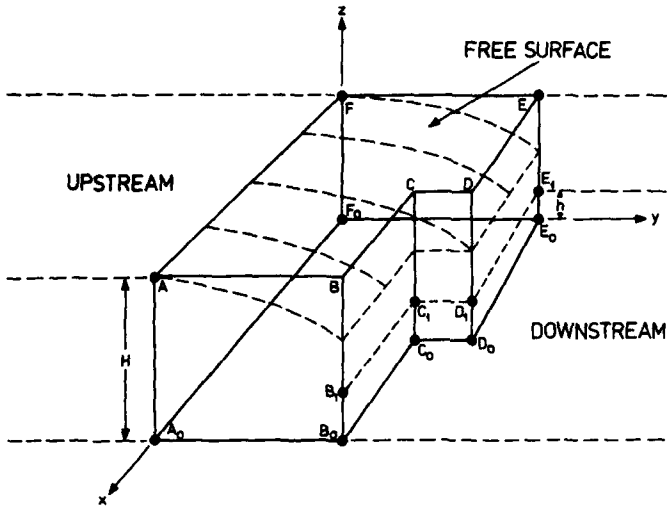


FIG. 4.1. Flow through a three-dimensional porous dam with L-shaped horizontal cross-section.

Here $\alpha(x,y)$ is the solution of the two-dimensional mixed boundary value problem

(a) $$\alpha_{xx} + \alpha_{yy} = 0, \quad \text{in} \quad A_0B_0C_0D_0E_0F_0 ,$$

(b) $$\alpha = \begin{cases} \frac{1}{2} H^2, & \text{on} \quad A_0F_0 \\ \frac{1}{2} h^2, & \text{on} \quad B_0C_0D_0E_0 \end{cases} \tag{4.7}$$

(c) $$\alpha_x \equiv \alpha_n = 0, \quad \text{on} \quad A_0B_0 \cup E_0F_0 .$$

To solve the problem (4.4)-(4.7) numerically we introduce a grid with $\Delta x = \Delta y = \Delta z$ and denote the approximation to $u([i - 2]\Delta x,[j - 1]\Delta y,[k - 1]\Delta z)$ by w_{ijk}, and the approximation to $\alpha([i - 2]\Delta x,[j - 1]\Delta y)$ by $w_{ij} \equiv w_{ij1}$, for $2 \le i \le M - 1$ and $1 \le j \le N$. As in Bruch [1980] the computation proceeds in two stages:

Stage I: The two-dimensional problem (4.7) is approximated by replacing the differential equation (4.7a) by the difference equations

$$4w_{ij} - w_{i+1,j} - w_{i-1,j} - w_{i,j+1} - w_{i,j-1} = 0 \ . \tag{4.8}$$

The Dirichlet boundary conditions (4.7b) are satisfied by computing and storing the values of $w_{ij1} = \alpha_{ij}$ on A_0F_0 and $B_0C_0D_0E_0$. The Neumann conditions (4.7c) are satisfied by introducing two fictitious rows of gridpoints, adjacent to A_0B_0 and E_0F_0 respectively, and requiring that the values of w on a fictitious row should be equal to the values of w on the corresponding interior row; that is, $w_{1j} = w_{3j}$ and $w_{M,j} = w_{M-2,j}$, for $1 \leq j \leq N$.

The resulting system of equations is solved using a simple modification of the subroutine PROJSOR (see Figure 3.5): the term -GRID2 is dropped from statement number 3; statement number 5 is deleted; and the statements

$$\begin{aligned} W(1, \ ) &= W(3, \ ) \ , \\ W(M, \ ) &= W(M - 2, \ ) \ , \end{aligned} \tag{4.9}$$

are inserted between statements number 1 and 2, so as to make the values at the fictitious points equal to the corresponding interior values;

Stage II: The three-dimensional problem (4.4) is approximated by the LCP

(a) $$6w_{i,j,k} \geq w_{i+1,j,k} + w_{i-1,j,k} + w_{i,j+1,k} + w_{i,j-1,k} + w_{i,j,k-1} + w_{i,j,k+1} - (\Delta x)^2 \ ,$$

(b) $$w_{i,j,k} \geq 0 \ , \tag{4.10}$$

(c) $$w_{i,j,k}[6w_{i,j,k} - w_{i+1,j,k} - w_{i-1,j,k} - w_{i,j+1,k} - w_{i,j-1,k} - w_{i,j,k+1} - w_{i,j,k-1} + (\Delta x)^2] = 0 \ .$$

The Dirichlet boundary conditions (4.5) are readily imposed, while the Neumann conditions (4.6) are treated by introducing fictitious sides parallel to the sides ABB_0A_0 and EFF_0E_0, and requiring that the values of w on the fictitious sides be equal to the values of w at the corresponding interior points.

To solve the LCP (4.10) we introduce a three-dimensional red-black partitioning of the gridpoints, so that each red(black) gridpoint has six black(red) neighbors. (It should be noted that the red/black ordering on any horizontal plane is the negation of the red/black orderings on the adjacent horizontal planes.) As in the two-dimensional problem treated in Section 3, each projected SOR iteration can be broken down into two stages: a red stage in which projected SOR is applied to all the red points in the three-dimensional w array, followed by a similar black stage. In detail:

Red Stage

$$\text{(a)} \qquad z_{ijk}^{(k,red)} = w_{i+1,j,k}^{(k,black)} + w_{i-1,j,k}^{(k,black)} + w_{i,j+1,k}^{(k,black)} + w_{i,j-1,k}^{(k,black)} + w_{i,j,k+1}^{(k,black)} + w_{i,j,k-1}^{(k,black)} - (\Delta x)^2 ,$$

$$\text{(b)} \qquad w_{ijk}^{(k+\frac{1}{2},red)} = (\omega/6) z_{ijk}^{(k,red)} + (1-\omega) w_{ijk}^{(k,red)} , \qquad (4.11)$$

$$\text{(c)} \qquad w_{ijk}^{(k+1,red)} = \max\{0, w_{ijk}^{(k+\frac{1}{2},red)}\} .$$

Black Stage

$$\text{(a)} \qquad z_{ijk}^{(k,black)} = w_{i+1,j,k}^{(k+1,red)} + w_{i-1,j,k}^{(k+1,red)} + w_{i,j+1,k}^{(k+1,red)} + w_{i,j-1,k}^{(k+1,red)} + w_{i,j,k+1}^{(k+1,red)} + w_{i,j,k-1}^{(k+1,red)} - (\Delta x)^2 ,$$

$$\text{(b)} \qquad w_{ijk}^{(k+\frac{1}{2},black)} = (\omega/6) z_{ijk}^{(k,black)} + (1-\omega) w_{ijk}^{(k,black)} , \qquad (4.12)$$

$$\text{(c)} \qquad w_{ijk}^{(k+1,black)} = \max\{0, w_{ijk}^{(k+\frac{1}{2},black)}\} .$$

To implement the algorithm (4.11), (4.12) it was assumed that the dimensions of Ω_2 were such that the gridpoints on any horizontal cross-section of the dam could be regarded as a subset of a 32 × 32 array. The solution w was stored as an array of matrices, the matrix $W(\ ,\ ,k)$ containing the values of w on the horizontal

plane at a height $(k - 1)\Delta z$. To control the parallel computation two logical matrices were used: MASKRB which is true at interior red gridpoints in the current horizontal cross-section and false otherwise; and MASKMASK which is true at interior points of Ω_2 and false otherwise.

The algorithm (4.11), (4.12) was implemented in two ways:

Implementation 1:

During each red(black) stage the horizontal planes were updated in turn, and on each plane the red(black) points were updated in parallel.

The computation of $z^{(k)}$ requires five additions and one subtraction. Given an unlimited number of processors, n additions/subtractions require $\log_2 n$ steps, so that six additions/subtractions require at least three steps. By taking advantage of idle PEs, and remembering that, on the DAP, shift operations are much faster than arithmetic operations, the DAP-FORTRAN subroutine in Figure 4.2 is an efficient implementation of (4.11), (4.12) (compare Figure 3.5). A full listing of the program is given in Appendix C.

The subroutine in Figure 4.2 uses the functions SHS(outh) and SHN(orth) to shift W instead of the equivalent, but slower, statements (4.9).

Implementation 2:

As in the three-dimensional magnetohydrodynamic code of Reddaway [1976] we rearrange the values of w. The horizontal planes are considered in pairs, and the red points on each even-numbered plane are exchanged with the corresponding black points on the next odd-numbered plane. As a result, instead of having n planes, each containing red and black points in a checkerboard pattern, we have n/2 planes of red points interleaved with n/2 planes of black points. This makes it possible to use simultaneously all interior PEs for arithmetic.

The corresponding subroutine is given in Figure 4.3, and a full listing of the program is given in Appendix D. To save time the test for convergence is executed only every TIMES iterations, where TIMES is an input parameter.

The subroutine in Figure 4.3 assumes that there is an even number of planes. To avoid additional testing, it is assumed that a copy of the top plane is stored above the top plane.

The two implementations were run on the problem with $H = 10$, $h = 0$, $AF = FE = 20$, $CD = BC = 10$, which was chosen because it had previously been solved by Bruch [1980]. For comparison, the problem was also solved on the UNIVAC 1180 using single precision arithmetic and optimized FORTRAN code. The measured computation times per projected SOR iteration (including both red and black stages) were:

Implementation 1:	32ms
Implementation 2:	16.0-18.2ms (dependent on frequency of convergence tests)
UNIVAC 1180:	34ms

so that implementation 2 on the Pilot DAP is about 2 times faster than the UNIVAC 1180.

The estimated time per SOR iteration (implementation 2) was found as in Figure 3.6, and was found to lie between 15.4 and 17.4ms, depending upon the frequency of convergence tests.

For this problem only 383 (i.e. $21 \times 23 - 10 \times 10$) of the 1024 PEs were used.

```
C   THE MAIN LOOP - PROCESS ALL THE  Z  PLANES
C
          SUBROUTINE MAIN LOOP
          COMMON /ISCA/ TOPPLANE,M
          COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
          INTEGER TOPPLANE,M
          INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
          REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
          COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
          COMMON /RMAT/W( ,,25)
          COMMON /SUBLMAT/MASKRB ( , ), MASKMASK( , )
          LOGICAL MASKRB, MASKMASK
          REAL SAVEW( , ), Z ( , ), Z1( , )
          REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
          INTEGER NUMBTIMES, TOPPLANE
          LOGICAL TEMPMASK( , ) , DONE, WSIGN( , )
          EQUIVALENCE (WSIGN,Z)
          ALPHA = OMEGA * 1.0 / 6.0
          BETA = 1.0 - OMEGA
C
C   WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
          NUMBITERS = 0
          WIDTHGRID = 1.0
          WIDGRID2 = WIDTHGRID * WIDTHGRID
C
C   SAVE THE MASKRB FOR LATER RESTORATION
C
          TEMPMASK = MASKRB
C
C   MAXDIFF IS THE MAXIMUM DIFFERENCE BETWEEN SAVEW AND  W( ,,K)  AFTER  W( ,,K)
C   HAS ITS RED (OR BLACK) VALUES CHANGE (FOR ALL  K)
C
1         MAXDIFF = 0.0
          NUMBITERS = NUMBITERS + 1
          MASKRB = TEMPMASK
          DO  30  NUMBTIMES = 1,2
C
C   ITERATE FROM THE 2ND PLANE TO THE TOP PLANE
C
          DO  20  K = 2,  TOPPLANE
          SAVEW = W( ,,K)
C
C   REVERSE RED/BLACK FOR SUCCESSIVE PLANES
C
          MASKRB(MASKMASK) = .NOT. MASKRB
C
C   SUM THE SIX NEIGHBORS
          SAVEW(1, ) = SHN(SAVEW,2)
          SAVEW(M, ) = SHS(SAVEW,2)
          Z = SAVEW(-,-)
          Z1 = SAVEW
          Z1(MASKRB) = W( ,,K + 1)
          Z(MASKRB) = W( ,,K - 1)
          Z = Z + Z1
          Z1 = Z( ,+)
```

```
            Z1(.NOT.MASKRB) = -WIDGRID2
            Z = Z + Z1
            Z = Z + Z(+, )
C
C     STORE THE AVERAGE OF THE SIX NEIGHBORS IN  W  ONLY IN THE RED (OR BLACK) CELLS
C
            Z = ALPHA * Z + BETA * SAVEW
            Z(WSIGN) = 0.0
            W(MASKRB,K) = Z
C
C     FIND THE MAXIMUM DIFFERENCE ON THIS PLANE
C
            MAXSOFAR = MAX(ABS(SAVEW - Z),MASKRB)
            IF  (MAXSOFAR .GT. MAXDIFF) MAXDIFF = MAXSOFAR
20          CONTINUE
C
C     REVERSE STATE OF ORIGINAL MASKRB FOR THE 2ND PASS THROUGH THE PLANES
C
            MASKRB(MASKMASK) = .NOT. TEMPMASK
30          CONTINUE
            DONE = (NUMBITERS.GT.DAMMAXITERS).OR.(MAXDIFF.LE.DAMEPSILON)
            IF (.NOT. DONE) GOTO 10
            MASKRB = TEMPMASK
            RETURN
            END
C
C
```

FIG. 4.2. First implementation of (4.11) and (4.12).

```
      THE MAIN LOOP - PROCESS ALL THE  Z  PLANES

          SUBROUTINE  MAIN  LOOP
          COMMON /ISCA/ TOPPLANE,M
          COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
          INTEGER TOPPLANE,M
          INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
          REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
          COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
          COMMON /RMAT/W( ,,25)
          COMMON /SUBLMAT/MASKRB( , ), MASKMASK( , )
          COMMON /WORK/ Z,WK
          LOGICAL MASKRB, MASKMASK
          REAL SAVEW( , ), Z( , ), Z1(, ), WKP1( , ), WK( , ), MAXD( , )
          REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
          INTEGER NUMBTIMES, TOPPLANE
          LOGICAL TEMPMASK( , ), DONE, WSIGN( , ), TEST, NOTTEST
          EQUIVALENCE (WSIGN,Z), (Z,Z1), (WK,WKP1)

          ALPHA = OMEGA * 1.0 / 6.0
          BETA = 1.0 - OMEGA
C
C     WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
          NUMBITERS = 0
          WIDTHGRID = 1.0
          WIDGRID2 = WIDTHGRID * WIDTHGRID
C
C
1         TEST = TIMES .EQ. 1
          NOTTEST = .NOT. TEST
          ITIMES = TIMES
          MAXD = 0.0
C
C
C     ALTER ALL THE ODD NUMBERED PLANES:
2         KM2 = 1
          DO 20 K = 2,TOPPLANE,2
          SAVEW = W( ,,K + 1)
          WK = W( ,,K)
          WK(1, ) = SHN(WK,2)
          WK(M, ) = SHS(WK,2)
          Z = WK + WK(-,-)
          Z = (Z(+, ) + Z( ,+) + WK + MERGE(W( ,,KM2),W( ,,K + 2),MASKRB)
             - WIDGRID2) * ALPHA + SAVEW * BETA
          Z(WSIGN) = 0.0
          W(MASKMASK,K + 1) = Z
          IF (NOTTEST) GOTO 20
          Z1 = ABS(SAVEW - Z)
          MAXD(Z1.GT.MAXD) = Z1
20        KM2 = K
C
C
```

```
C    ALTER ALL THE EVEN NUMBERED PLANES:
         DO 21 K = 2,TOPPLANE,2
         SAVEW = W( ,,K)
         WKP1 = W( ,,K + 1)
         WKP1(1, ) = SHN(WKP1,2)
         WKP1(M, ) = SHS(WKP1,2)
         Z = WKP1 + WKP1(-,-)
         Z = (Z(+, ) + Z( ,+) + WKP1 + MERGE(W( ,,K + 3),W( ,,K - 1),MASKRB)
            - WIDGRID2) * ALPHA + SAVEW * BETA
         Z(WSIGN) = 0.0
         W(MASKMASK,K) = Z
         IF (NOTTEST) GOTO 21
         Z1 = ABS(SAVEW-Z)
         MAXD(Z1.GT.MAXD) = Z1
21       CONTINUE
C
C
         ITIMES = ITIMES - 1
         IF (ITIMES.GT.1) GOTO 2
         IF (ITIMES.EQ.0) GOTO 3
         TEST = .TRUE.
         NOTTEST = .FALSE.
         GOTO 2
C
3        NUMBITERS = NUMBITERS + TIMES
         MAXDIFF = MAX(MAXD,MASKMASK)
         IF (MAXDIFF.LT.DAMEPSILON) GOTO 4
C
         IF (NUMBITERS.LT.DAMMAXITERS) GOTO 1
4        RETURN
         END
```

FIG. 4.3. Second implementation of (4.11) and (4.12).

5. Future possibilities

(a) For purposes of comparison we have used previously published problems but they have dimensions which do not match the DAP array closely. In many practical problems the resolution would be tailored to the DAP dimensions to achieve higher performance.

(b) The programs presented are readily extensible to larger problems on correspondingly larger DAPs such as the production 64 × 64; it is only necessary to change the boundaries. The time to process one plane would be unchanged.

(c) Performance on small three-dimensional problems can be improved by mapping several problem planes onto one DAP matrix.

(d) Problems with large "horizontal" dimensions can be mapped with each PE holding a small neighborhood group of points. Performance improves because each PE holds both black and red points (Hunt [1979]).

(e) Very large problems cannot be held entirely within DAP store. For example with four times as many points in a horizontal plane as there are PEs the limit is about 26 planes with 4K bits per PE or about 122 planes with 16K bits per PE. With backing store the transfer rates with N active problem planes in the DAP can be minimized by advancing each plane $(N - 2)/2$ iterations per backing store fetch. Hence it should be possible to achieve a balance between input-output and processing times (Reddaway [1976]).

(f) Problems of this type offer possibilities for using fixed point arithmetic (with suitable scaling) and using low precision for computing the iterative corrections. This is much faster than floating point work and performance improvements as large as a factor of 10 are predicted without loss of accuracy in the final solution.

6. Conclusions

We have demonstrated that two- and three-dimensional linear complementarity problems can be solved on DAP with high performance and easy programming using a version of projected SOR. There is scope for even higher performance and for tackling a wide range of problem sizes.

Acknowledgements

Two authors, CWC and JS, gratefully acknowledge the provision by International Computers Ltd. of facilities for using the pilot DAP at Stevenage, England.

REFERENCES

Baiocchi, C.: Su un problema di frontiera libera connesso a questioni di idraulica. Ann. Mat. Pura Appl. (4) 92, 107-127 (1972).

Balinski, M. L. and Cottle, R. W. (editors): Complementarity and Fixed Point Problems. Amsterdam: North Holland, 1978.

Brandt, A. and Cryer, C. W.: Multigrid algorithms for the solution of linear complementarity problems arising from free boundary problems. Technical Summary Report #2131, Mathematics Research Center, University of Wisconsin-Madison, 1980.

Bruch, J. C., Jr.: A survey of free boundary value problems in the theory of fluid flow through porous media: variational inequality approach - Part II. Advances in Water Resources 3, 115-124 (1980).

Cottle, R. W., Giannessi F. and Lions, J. L. (editors): Variational Inequalities and Complementarity Problems. New York: John Wiley, 1980.

Cottle, R. W., Golub, G. H. and Sacher, R. S.: On the solution of large structured linear complementarity problems: the block partitioned case. Appl. Math. Optim. 4, 347-363 (1978).

Cryer, C. W.: The solution of a quadratic programming problem using systematic overrelaxation. SIAM J. Control 9, 385-392 (1971).

Cryer, C. W.: Successive overrelaxation methods for solving linear complementarity problems arising from free boundary problems. Proceedings, Seminar on Free Boundary Problems, Pavia, 1979, E. Magenes (editor), vol. 2, pp. 109-131. Instituto Nazionale di Alta Matematica Francesco Severi, Rome 1980.

Cryer, C. W. and Dempster, M. A. H.: Equivalence of linear complementarity problems and linear programs in vector lattice Hilbert spaces. SIAM J. Control 18, 76-90 (1980).

Duvaut, G. and Lions, J. L.: Inequalities in Mechanics and Physics. Paris: Dunod, 1976.

Flanders, P. M.: FORTRAN extensions for a highly parallel processor. In Supercomputers. London: Infotech International Ltd., 1979.

Flanders, P. M., Hunt, D. J., Reddaway, S. F. and Parkinson, D.: Efficient high speed computing with the distributed array processor. In High Speed Computer and Algorithm Organization, Kuck, D. J. (editor). New York: Academic Press, 1977.

France, P. W.: Finite element analysis of three-dimensional groundwater flow problems. J. of Hydrology 21, 381-398 (1974).

Glowinski, R.: La methode de relaxation. Rendiconti di Mathematica 14, Universita di Roma, 1971.

Gostick, R. W.: Software and algorithms for distributed-array processors. ICL Technical J. 1, 116-135 (1979).

Heller, D.: A survey of parallel algorithms in numerical linear algebra. SIAM Review 20, 740-777 (1978).

Hunt, D. J.: Numerical solution of Poisson's equation on an array processor using iterative techniques. Report No. CM21, International Computers Limited, Research and Advanced Development Centre, Stevenage, 1974.

Hunt, D. J: Application techniques for parallel hardware. In Supercomputers. London: Infotech International Ltd., 1979.

ICL Technical Publication #6918, DAP Fortran Language 1979.

Kinderlehrer, D. and Stampacchia, G.: An Introduction to Variational Inequalities and their Applications. New York: Academic Press, 1980.

Mangasarian, O. M.: Solution of symmetric linear complementarity problems by iterative methods. J. Optimization Theory and Applic. 22, 465-485 (1977).

Reddaway, S. F.: A 3D magnetohydrodynamics code (3DMHD) on DAP. Report No. CM59, International Computers Limited, Research and Advanced Development Centre, Stevenage, November 1976.

Reddaway, S. F.: The DAP approach. In Supercomputers. London: Infotech International Ltd., 1979.

Stampacchia, G.: On the filtration of a fluid through a porous medium with variable cross-section. Russian Math. Surveys 29, 89-102 (1974).

CWC:PMH:SFR:JS:scr

APPENDIX A: The Pilot Host-DAP Interface.

In the Pilot DAP system the store of the DAP is not an integral part of the host's store as with the production DAP's. It is therefore necessary to explicitly move data between the host and DAP and this is achieved by using standard host FORTRAN subroutines. The subroutine names begin with DAPTO or DAPFROM depending on whether they move data into or out of the DAP. The remaining letters of the name indicate the type (integer or real denoted by I or E) and rank (scalar, vector or matrix denoted by S, V, or M) of the variable transferred. Parameters of DAPTO and DAPFROM give the name of the host program variable and the location within the DAP in terms of the name of the common area and the offset from the start of this area.

Initiation of DAP processing is also less direct on the Pilot system with DAP-FORTRAN subroutines being called via the standard host FORTRAN subroutine DAPGO. A statement of the form:

```
CALL DAPGO('DAPSUB',N)
```

will suspend execution of the host FORTRAN and transfer control to the DAP-FORTRAN subroutine DAPSUB. Execution of the host FORTRAN is resumed after DAPSUB and any further levels of DAP-FORTRAN subroutines have been executed. The parameter N gives the maximum number of seconds allowed for DAP processing.

APPENDIX B: The two-dimensional dam problem.

```
      MASTER ECALPAL
C
C     THIS PROGRAM USES FINITE DIFFERENCES TO SOLVE NUMERICALLY
C     THE VARIATIONAL INEQUALITY FLOW THROUGH A RECTANGULAR DAM,
C
C     THE ORIGINAL PROGRAM WAS WRITTEN BY DR  C, CRYER AND WAS
C     EXECUTED ON A UNIVAC 1110 AT THE UNIVERSITY OF WISCONSIN
C     (MARCH 24, 1977; JUNE 1979)
C
C     THIS PROGRAM WAS WRITTEN BY JOHN STANSBURY IN DAP FORTRAN
C     AND EXECUTED ON AN ICL 1900 (HOST) AND ON THE DAP AT THE
C     RESEARCH AND ADVANCED DEVELOPMENT CENTRE IN STEVENAGE
C     ENGLAND (JUNE 17=20, 1979).
C
C                         EXPLAINATION OF VARIABLES
C     W - THE VALUES AT THE GRIDPOINTS
C     Z - THE NEW VALUES AT THE GRIDPOINTS
C     MASK - USED TO IMPLEMENT THE RED/BLACK SCHEME

C     MASK MASK - USED IN SETTING UP THE MASK, AND IN SWITCHING STATES
C     ACC ERRS - DIFFERENCES BETWEEN THE OLD AND THE NEW VALUES
C
C     OMEGA - THE OVERRELAXATION PARAMETER
C     EPSILON - THE DESIRED ACCURACY
C     DONE - A TEMPORARY LOGICAL VARIABLE
C     NUMB ITERATIONS - THE NUMBER OF ITERATIONS
C     MAX ITERATIONS - THE MAXIMUM NUMBER OF ITERATIONS
C     NUMB ROWS - THE NUMBER OF ROWS IN THE GRID
C     NUMB COLS - THE NUMBER OF COLUMNS IN THE GRID
C     WIDTH GRID - WIDTH OF ONE UNIT IN THE GRID
C     HEIGHT GRID - HEIGHT OF ONE UNIT IN THE GRID
C     DAM WIDTH - SELF EXPLANATORY
C     DAM HEIGHT - SELF EXPLANATORY
C     X GRIDPOINTS - THE NUMBER OF GRIDPOINT  IN THE X DIRECTION
C     Y GRIDPOINTS - THE NUMBER OF GRIDPOINT  IN THE Y DIRECTION
C     NOTE: IT IS ASSUMED THAT HEIGHT GRID = WIDTH GRID
C
C     DECLARATION OF VARIABLES
      COMMON /RMAT/W(32,32)
      COMMON /RSCA/MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
      COMMON /ISCA/MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS
     1     X GRIDPOINTS,Y GRIDPOINTS
      REAL W, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT, MAX DIFF
      INTEGER MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS,
     1     X GRIDPOINTS, Y GRIDPOINTS
      PAUSE 99
C
C     READ IN DIMENSIONS OF THE DAM, AND THE MAXIMUM NUMBER OF ITERATIONS
1     READ(5,100,END=20) DAM WIDTH, DAM HEIGHT, MAX ITERATIONS
100   FORMAT(2F10.5, I5)
      IF((DAM WIDTH .GT. 1.0) .AND. (DAM HEIGHT .GT. 1.0)) GOTO 5
20    WRITE(6,110) DAM WIDTH, DAM HEIGHT
110   FORMAT(' DAM WIDTH = ',F10.5,8H   NY = ,F10.5)
      PAUSE 98
```

```
C
C    READ IN THE NUMBER OF ROWS AND COLUMNS IN THE GRID, AND EPSILON
5      READ(5,120) NUMB COLS, NUMB ROWS, EPSILON
120    FORMAT(2I5, F10.5)
       WRITE(6,130) DAM HEIGHT, DAM WIDTH, EPSILON, NUMB ROWS,
      1      NUMB COLS, MAX ITERATIONS
130    FORMAT(17H1INPUT PARAMETERS,/14H DAM HEIGHT = ,E13.6,
      1      13H DAM WIDTH = ,E13.6,11H EPSILON = ,E13.6,/
      2      10H NUMBER OF ROWS = ,I5,21H NUMBER OF COLUMNS = ,I5,
      3      22H MAXIMUM ITERATIONS = ,I5)
C
C    CONVERT AND TRANSFER DATA TO DAP
       CALL DAPTO ES(EPSILON,4HRSCA,2)
       CALL DAPTO ES(DAM WIDTH,4HRSCA,3)
       CALL DAPTO ES(DAM HEIGHT,4HRSCA,4)
       CALL DAPTO IS(MAX ITERATIONS,4HISCA,0)
       CALL DAPTO IS(NUMB ROWS,4HISCA,2)
       CALL DAPTO IS(NUMB COLS,4HISCA,3)
C
C    CALL MAIN DAP FORTRAN SUBROUTINE
       CALL DAPGO(7HLAPLACE,10)
C
C    CONVERT AND TRANSFER DAP DATA BACK TO  900
       CALL DAPFROM EM(W,4HRMAT,0)
       CALL DAPFROM ES(OMEGA,4HRSCA,1)
       CALL DAPFROM ES(MAX DIFF,4HRSCA,0)
       CALL DAPFROM IS(NUMB ITERATIONS,4HISCA,1)
       CALL DAPFROM IS(Y GRIDPOINTS, 4HISCA, 5)
       CALL DAPFROM IS(X GRIDPOINTS, 4HISCA, 4)
C
C    WRITE OUT THE SCALERS
       WRITE(6,140) NUMB ITERATIONS, OMEGA, MAX DIFF
140    FORMAT(' ',///,' NUMBER OF ITERATIONS = ',I5,' OMEGA = ',E13.6,
      1    ' MAXIMUM DIFFERENCE = ',E13.6)
C
C    WRITE OUT W
       J = Y GRIDPOINTS
10     WRITE(6,150) (W(I,J), I=1,X GRIDPOINTS)
150    FORMAT(' ',9E13.6)
       J = J - 1
       IF (J .GE. 1) GOTO 10
       WRITE(6,170)
170    FORMAT(' ',///)
       DO 30 J = 1, Y GRIDPOINTS
       WRITE(6,160) (W(I,J), J = 1, X GRIDPOINTS)
160    FORMAT(' ',9E13.6)
30     CONTINUE
       GOTO 1
       END
```

```
      SUBROUTINE LAPLACE
C
C     SUBROUTINE TO CARRY OUT THE CALCULATIONS ON THE DAP
        COMMON /RMAT/W(,)
        COMMON /RSCA/MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
        COMMON /ISCA/MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS
        COMMON /ISCA/X GRIDPOINTS, Y GRIDPOINTS
        REAL W, MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
        INTEGER MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS
        INTEGER X GRIDPOINTS, Y GRIDPOINTS
C
C     SET UP DAP FORTRAN COMMON AREAS
        COMMON /SUBRSCA/HEIGHT GRID, WIDTH GRID
        COMMON /SUBLMAT/MASK(,), MASK MASK(,)
        REAL HEIGHT GRID, WIDTH GRID
        LOGICAL MASK, MASK MASK
C
C     INITILIZE ALL VARIABLES

        CALL INIT OTHERS
        CALL INIT MASK
        CALL INIT W
C
C     PERFORM THE ACTUAL CALCULATIONS
        CALL MAIN LOOP
C
C     WE HAVE NOW EITHER ACHEIVED THE DESIRED ACCURACY (I.E., MAX ERR
C     <= EPSILON) OR WE HAVE ITERATED MORE THAN MAX ITERATIONS TIMES.
        RETURN

      SUBROUTINE INIT MASK
C
C     SUBROUTINE TO INITIALIZE THE MASK
        COMMON /ISCA/MAX ITERATIONS, NUMB ITERATIONS,NUMB ROWS,NUMB COLS
        COMMON /ISCA/X GRIDPOINTS, Y GRIDPOINTS
        COMMON /SUBLMAT/MASK(,), MASK MASK(,)
        INTEGER MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS
        INTEGER X GRIDPOINTS, Y GRIDPOINTS
        LOGICAL MASK, MASK MASK
C
C     MASK MASK IS A LOGICAL MASK USED IN SETTING UP AND IN CHANGING THE
C     STATE OF THE MASK, MASK MASK(I,J) IS FALSE WHERE I = 1 OR I >
C     NUMB ROWS; AND WHERE J = 1 OR WHERE J > NUMB COLS, IT IS TRUE
C     ELSEWHERE,

        MASK MASK = .NOT. (ALTC(NUMB ROWS) .OR. ALTR(NUMB COLS))
        MASK MASK(1,) = .FALSE.
        MASK MASK(,1) = .FALSE.
C
C     MASK IS MUCH THE SAME AS MASK MASK, EXCEPT THAT WHERE MASK MASK
C     IS TRUE, MASK ALTERNATES BETWEEN TRUE AND FALSE, DEPENDING
C     ON THE FOLLOWING:
C        IF MASK MASK(I,J) = TRUE AND I + J IS EVEN THEN MASK(I,J)
C        = TRUE, ELSE, MASK(I,J) = FALSE.
C
        MASK = ALTC(1) .LEQ. ALTR(1)
        MASK(,NOT.MASK MASK) = .FALSE.
        TRACE 127 (MERGE(1,0,MASK MASK))
        TRACE 127 (MERGE(1,0,MASK))
        RETURN
```

```
      SUBROUTINE INIT OTHERS
C
C     SUBROUTINE TO INITIALIZE THE OTHER VARIABLES
        COMMON /RSCA/MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
        COMMON /ISCA/MAX ITERATIONS,NUMB ITERATIONS,NUMB ROWS,NUMB COLS
        COMMON /ISCA/X GRIDPOINTS, Y GRIDPOINTS
        COMMON /SUBRSCA/HEIGHT GRID, WIDTH GRID
        REAL MAX DIFF, OMEGA, EPSILON, DAM WIDTH, DAM HEIGHT
        REAL HEIGHT GRID, WIDTH GRID
        INTEGER X GRIDPOINTS, Y GRIDPOINTS, MAX ITERATIONS, NUMB ITERATIONS
        INTEGER NUMB ROWS, NUMB COLS
C
C     FOR THE TEST RUN,
C        DAM HEIGHT = 24.0
C        DAM WIDTH = 16.0

        X GRIDPOINTS = NUMB COLS + 1
        Y GRIDPOINTS = NUMB ROWS + 1
        HEIGHT GRID = DAM HEIGHT / NUMB ROWS
        WIDTH GRID = DAM WIDTH / NUMB COLS
        OMEGA = 1.8
        NUMB ITERATIONS = 0
        RETURN
      SUBROUTINE INIT W
        COMMON /RSCA/MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
        REAL MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
        COMMON /ISCA/MAX ITERATIONS,NUMB ITERATIONS,NUMB ROWS,NUMB COLS
        COMMON /ISCA/X GRIDPOINTS, Y GRIDPOINTS
        COMMON /SUBRSCA/HEIGHT GRID, WIDTH GRID
        REAL HEIGHT GRID, WIDTH GRID
        INTEGER MAX ITERATIONS, NUMB ITERATIONS,NUMB ROWS, NUMB COLS
        INTEGER X GRIDPOINTS, Y GRIDPOINTS
        COMMON /RMAT/W(,)
        REAL W, TEMPV(), TEMPS
        REAL INDEX()
C
C     INITIALIZE THE W MATRIX, I.E., SET THE CONSTANT BOUNDARY VALUES
C     AND THE INITIAL NON-BOUNDARY VALUES.
C
C     SET UP A VECTOR CONTAINING THE INDICES

        DO 30 I = 1,32
        INDEX(I) = I - 1.0
30      CONTINUE
C
C     SET ALL OF W = ZERO
        W = 0.0
C
C     CALCULATE TWO TEMPORARY CONSTANTS
        TEMPS = DAM HEIGHT * DAM HEIGHT * 0.5
        TEMPV = INDEX / NUMB ROWS
        W(,1) = TEMPS * (1.0 - INDEX * WIDTH GRID / DAM WIDTH)
        W(1,) = TEMPS * (1.0 - TEMPV) * (1.0 - TEMPV)
        TRACE 127 (W)
        TRACE 127 (TEMPS, DAM HEIGHT, WIDTH GRID)
        TRACE 127 (DAM WIDTH, HEIGHT GRID)
        TRACE 127 (TEMPV)
        TRACE 127 (INDEX)
        RETURN
```

```
      SUBROUTINE MAIN LOOP
C
C     SUBROUTINE TO CARRY OUT THE CALCULATIONS
        COMMON /RMAT/W(,)
        COMMON /RSCA/MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
        COMMON /ISCA/MAX ITERATIONS,NUMB ITERATIONS,NUMB ROWS,NUMB COLS
        COMMON /ISCA/X GRIDPOINTS, Y GRIDPOINTS
        COMMON /SUBLMAT/MASK(,), MASK MASK(,)
        COMMON /SUBRSCA/HEIGHT GRID, WIDTH GRID
        REAL W,MAX DIFF,OMEGA,EPSILON,DAM WIDTH,DAM HEIGHT
        REAL HEIGHT GRID, WIDTH GRID
        LOGICAL MASK, MASK MASK
        INTEGER X GRIDPOINTS, Y GRIDPOINTS
        INTEGER MAX ITERATIONS, NUMB ITERATIONS, NUMB ROWS, NUMB COLS
C
C     LOCAL VARIABLES
        REAL Z(,), WID GRID 2, Z MIN W(,), SAVE W(,)
        INTEGER NUMB TIMES
        LOGICAL DONE, W SIGN(,)
        EQUIVALENCE (W SIGN, W)
C
C     ENSURE THAT W IS >= ZERO EVERYWHERE

        W(WSIGN) = 0.0
C
C     CALCULATE THE CONSTANTS THAT ARE NEEDED LATER ON
        ALPHA = OMEGA * .25
        BETA = 1.0 - OMEGA
        WID GRID 2 = WIDTH GRID * WIDTH GRID
C
C     START MAIN LOOP
C     SAVE THE OLD VALUE OF W
40      SAVE W = W
        NUMB ITERATIONS = NUMB ITERATIONS + 1
        DO 45 NUMB TIMES = 1,2
C
C     REVERSE STATE OF MASK
        MASK(MASK MASK) = .NOT. MASK
C
C     CALCULATE Z ON ONLY THE RED (OR BLACK) POINTS AS DETERMINED BY
C     THE MASK
        Z = W * W(-,-)
        Z = Z(+,) + Z(,+) - WID GRID 2
        W(MASK) = ALPHA * Z + BETA * W
C
C     SPECIAL PART! ENSURE THAT W IS >= 0
        W(W SIGN .AND. MASK MASK) = 0.0
45      CONTINUE
C
C     FIND MAXIMUM DIFFERENCE BETWEEN OLD AND NEW W
        MAX DIFF = MAX(ABS(SAVE W - W))
C
C     CHECK TO SEE IF DESIRED ACCURACY IS ATTAINED, OR IF NUMBER OF
C     ITERATIONS HAS EXCEEDED LIMIT
        DONE=(MAX DIFF.LE.EPSILON).OR.(NUMB ITERATIONS.GT.MAX ITERATIONS)
        IF (.NOT. DONE) GO TO 40
C
        TRACE 125 (*)
```

```
      TRACE 625 (MAX DIFF)
C
C     KLUDGY ...
      IF(NUMB ITERATIONS.GT.MAX ITERATIONS)NUMB ITERATIONS=MAX ITERATION
C     ALL DONE MAIN LOOP
      RETURN
```

APPENDIX C: The three-dimensional dam problem - implementation 1.

```
      MASTER ECALPAL
C
C
      COMMON /RMAT/W(32,32)
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      INTEGER VI(32)
      REAL VR(32)
      EQUIVALENCE (VI(1),ZPLANES),(VR(1),DAMEPSILON)
      PAUSE 99
10    READ(2,100) DAMHEIGHT, DAMLFSIDE, DAMLBSIDE, DAMLBACK, DAMRBACK
      IF(DAMHEIGHT .LE. 0) PAUSE 00
      READ(2,110) MAXITERS, EPSILON
      DAMEPSILON = EPSILON
      BOTTOMEPSILON = EPSILON
      DAMMAXITERS = MAXITERS
      BOTTOMMAXITERS = MAXITERS
      NUMBITERS = 1
      OMEGA = 1.8
      ZPLANES = DAMHEIGHT + 1
      DAMFACE = DAMLBACK + DAMRBACK
      DAMRSIDE = DAMLFSIDE + DAMLBSIDE
      XPOINTS = DAMRSIDE + 1
      XFPOINTS = DAMLFSIDE + 1
      XBPOINTS = DAMLBSIDE
      YRPOINTS = DAMRBACK + 2
      YLPOINTS = DAMLBACK+1
      YPOINTS = YRPOINTS + YLPOINTS
      WRITE(6,120) DAMHEIGHT, DAMLFSIDE, DAMLBSIDE, DAMLBACK, DAMRBACK
       CALL DAPTO IV(VI,4HISCA,0)
      CALL DAPTO ES(DAMEPSILON,4HRSCA,0)
      CALL DAPTO ES(BOTTOMEPSILON,4HRSCA,1)
      CALL DAPTO ES(OMEGA,4HRSCA,2)
C
C     THE TIME OUT PERIOD IS SET TO 600 SECONDS
C
        CALL DAPGO(7HLAPLACE, 600)
        CALL DAPFROM EV(VR,4HRSCA,0)
        CALL DAPFROM IV(VI,4HISCA,0)
        WRITE(6,130) NUMBITERS, NUMBOT, MAXDIFF
```

```
C
C    WRITE OUT THE W MATRIX, FROM TOP TO BOTTOM PLANE
C
      NK = ZPLANES
      NI = YPOINTS
      NJ = XPOINTS
      DO 40 K = 1,NK
      WRITE (6,150) K
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      DO 40 I = 1,NI
      WRITE (6,140) (W(I,J),J=1,NJ)
40    CONTINUE
      GOTO 10
      DO 50 J = 1,NJ
      WRITE (6,160) J
      DO 50 KK = 1,NK
      K = NK-KK+1
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      WRITE (6,140) (W(I,J),I=1,NI)
50    CONTINUE
      DO 60 I = 1,NI
      WRITE (6,170) I
      DO 60 KK=1,NK
      K=NK-KK+1
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      WRITE (6,140) (W(I,J),J=1,NJ)
60    CONTINUE
      GOTO 10
100   FORMAT(5I0)
110   FORMAT(I0, E0.0)
120   FORMAT( 9H1HEIGHT =,I5,
     122H DAM LEFT FRONT SIDE =,I5,20HDAM LEFT BACK SIDE =,I5,
     212H LEFT BACK =,I5,12HRIGHT BACK =,I5,//)
130   FORMAT(23H NUMBER OF ITERATIONS =,I5,
     19H NUMBOT =,I5,20HMAXIMUM DIFFERENCE =,E13.6)
140   FORMAT(1H ,9E13.6)
150   FORMAT(///11H FOR PLANE ,I3)
170   FORMAT(///11H FOR SIDE  ,I3)
160   FORMAT(///11H FOR FACE  ,I3)
      END
```

```
C   START OF DAP-FORTRAN SECTION
C
C   SUBROUTINE TO CALL OTHER SUBROUTINES
C
      SUBROUTINE LAPLACE
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      CALL INIT MASK
      CALL INIT W
      CALL INIT BOTTOM PLANE
      CALL MAIN LOOP
      RETURN
      END
C
C   INITIALIZE THE MASKS
C
      SUBROUTINE INIT MASK
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      LOGICAL T(,)
C
C   MASKMASK IS TRUE IN THE AREA OF W(,,K) WHERE COMPUTATION TAKES PLACE
C
      MASKMASK = ROWS(2,YPOINTS-1) .AND. COLS(2,XPOINTS-1)
C
      T = .FALSE.
      T = ROWS(YRPOINTS, YPOINTS) .AND. COLS(XFPOINTS, XPOINTS)
      MASKMASK(T) = .FALSE.
C
C
C   MASKRB IS THE RED/BLACK SCHEME
C
      MASKRB = ALTC(1) .LEQ. ALTR(1)
      MASKRB(.NOT. MASKMASK) = .FALSE.
      TRACE 127 (MERGE(1,0,TRAN(MASKMASK)))
      TRACE 127 (MERGE(1,0,TRAN(MASKRB)))
      RETURN
      END
```

```
C     INITIALIZE THE W MATRIX
C
      SUBROUTINE INIT W
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      REAL TEMPS, TEMPS1
      TEMPS = DAMHEIGHT * DAMHEIGHT * 0.5
      DO 10 K = 1, ZPLANES
      W(,,K) = 0.0
      TEMPS1 = 1 - (K - 1) / EFLOAT(DAMHEIGHT)
      W(,1,K) = TEMPS * TEMPS1 * TEMPS1
10    CONTINUE
      RETURN
      END
C
C     INITIALIZE THE BOTTOM Z PLANE (I.E. W(,,1))
C
      SUBROUTINE INIT BOTTOM MATRIX
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      REAL Z(,), SAVEW(,), ALPHA, BETA
      LOGICAL DONE
      INTEGER NUMBTIMES
      NUMBOT = 0
      ALPHA = OMEGA * 0.25
      BETA = 1.0 - OMEGA
10    SAVEW = W(,,1)
      NUMBOT = NUMB OT + 1
      DO 45 NUMBTIMES = 1,2
      MASKRB(MASKMASK) = .NOT. MASKRB
      W(1,,1) = W(3,,1)
      W(YPOINTS,,1) = W(YPOINTS-2,,1)
      Z = W(,,1) + W(-,-,1)
      Z = Z(+,) + Z(,+)
      W(MASKRB,1) = ALPHA * Z + BETA * W(,,1)
```

```
45    CONTINUE
      MAXDIFF = MAX(ABS(SAVEW - W(,,1)))
      DONE=(MAXDIFF.LE.BOTTOMEPSILON).OR.(NUMBOT.GT.BOTTOMMAXITERS)
      IF (.NOT. DONE) GOTO 40
      TRACE 127(MAXDIFF, NUMBOT)
      RETURN
      END
C
C   THE MAIN LOOP - PROCESS ALL THE Z PLANES
C
      SUBROUTINE MAIN LOOP
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPTSM2, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RMAT/W(,,2)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      REAL SAVEW(,), Z(,), Z1(,)
      REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
      INTEGER NUMBTIMES, TOPPLANE
      LOGICAL TEMPMASK(,), DONE, WSIGN(,)
      EQUIVALENCE (WSIGN,Z)
      ALPHA = OMEGA * 1.0 / 6.0
      BETA = 1.0 - OMEGA
C
C   WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
      NUMBITERS = 0
      WIDTHGRID = 1.0
      WIDGRID2 = WIDTHGRID * WIDTHGRID
      YPTSM2 = YPOINTS - 2
      TOPPLANE = ZPLANES - 1
C
C   SAVE THE MASKRB FOR LATER RESTORATION
C
      TEMPMASK = MASKRB
C
C   MAXDIFF IS THE MAXIMUM DIFFERENCE BETWEEN SAVEW AND W(,,K) AFTER
C   W(,,K) HAS ITS RED (OR BLACK) VALUES CHANGE (FOR ALL K)
C
1     MAXDIFF = 0.0
      NUMBITERS = NUMBITERS + 1
      MASKRB = TEMPMASK
      DO 30 NUMBTIMES = 1,2
C
C   ITERATE FROM THE 2ND PLANE TO THE TOP PLANE
C
      DO 20 K = 2, TOPPLANE
      SAVEW = W(,,K)
C
C   REVERSE RED/BLACK FOR SUCCESSIVE PLANES
C
      MASKRB(MASKMASK) = .NOT. MASKRB
```

```
C
C     SUM THE SIX NEIGHBORS
C
      SAVEW(1,) = SHN(SAVEW,2)
      SAVEW(YPOINTS,) = SHS(SAVEW,2)
      Z = SAVEW(=,=)
      Z1 = SAVEW
      Z1(MASKRB) = W(,,K+1)
      Z(MASKRB) = W(,,K-1)
      Z = Z + Z1
      Z1 = Z(,+)
      Z1(.NOT.MASKRB) = WIDGRID2
      Z = Z + Z1
      Z = Z + Z(+,)
C
C     STORE THE AVERAGE OF THE SIX NEIGHBORS IN W ONLY IN THE
C     RED (OR BLACK) CELLS
C
      Z = ALPHA * Z + BETA * SAVEW
      Z(WSIGN) = 0.0
      W(MASKRB,K) = Z
C
C     FIND THE MAXIMUM DIFFERENCE ON THIS PLANE
C
      MAXSOFAR = MAX(ABS(SAVEW - Z),MASKRB)
      IF (MAXSOFAR .GT. MAXDIFF) MAXDIFF = MAXSOFAR
20    CONTINUE
C
C     REVERSE STATE OF ORIGINAL MASKRB FOR THE 2ND PASS THROUGH
C     THE PLANES
C
      MASKRB(MASKMASK) = .NOT. TEMPMASK
30    CONTINUE
      DONE = (NUMBITERS.GT.DAMMAXITERS).OR.(MAXDIFF.LE.DAMEPSILON)
      IF (.NOT. DONE) GOTO 10
      MASKRB = TEMPMASK
      RETURN
      END
```

APPENDIX D: The three-dimensional dam problem - implementation 2.

```
      MASTER ECALPAL
C
C
      COMMON /RMAT/W(32,32)
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      INTEGER VI(32)
      REAL VR(32)
      EQUIVALENCE (VI(1),ZPLANES),(VR(1),DAMEPSILON)
      PAUSE 99
10    READ(2,100) DAMHEIGHT, DAMLFSIDE, DAMLBSIDE, DAMLBACK, DAMRBACK
      IF(DAMHEIGHT .LE. 0) PAUSE 00
      READ(2,110) MAXITERS, EPSILON, TIMES
      DAMEPSILON = EPSILON
      BOTTOMEPSILON = EPSILON
      DAMMAXITERS = MAXITERS
      BOTTOMMAXITERS = MAXITERS
      NUMBITERS = 1
      OMEGA = 1.8
      ZPLANES = DAMHEIGHT + 1
      DAMFACE = DAMLBACK + DAMRBACK
      DAMRSIDE = DAMLFSIDE + DAMLBSIDE
      XPOINTS = DAMRSIDE + 1
      XFPOINTS = DAMLFSIDE + 1
      XBPOINTS = DAMLBSIDE
      YRPOINTS = DAMRBACK + 2
      YLPOINTS = DAMLBACK+1
      YPOINTS = YRPOINTS + YLPOINTS
      WRITE(6,120) DAMHEIGHT, DAMLFSIDE, DAMLBSIDE, DAMLBACK, DAMRBACK
       CALL DAPTO IV(VI,4HISCA,0)
      CALL DAPTO ES(DAMEPSILON,4HRSCA,0)
      CALL DAPTO ES(BOTTOMEPSILON,4HRSCA,1)
      CALL DAPTO ES(OMEGA,4HRSCA,2)
      CALL DAPTO IS(TIMES,4HRSCA,4)
C
C    THE TIME OUT PERIOD IS SET TO 600 SECONDS
C
      CALL DAPGO(7HLAPLACE, 600)
      CALL DAPFROM EV(VR,1HRSCA,0)
      CALL DAPFROM IV(VI,4HISCA,0)
      WRITE(6,130) NUMBITERS, NUMBOT, MAXDIFF
```

```
C
C     WRITE OUT THE W MATRIX, FROM TOP TO BOTTOM PLANE
C
      NK = ZPLANES
      NI = YPOINTS
      NJ = XPOINTS
      DO 40 K = 1,NK
      WRITE (6,150) K
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      DO 40 I = 1,NI
      WRITE (6,140) (W(I,J),J=1,NJ)
40    CONTINUE
      GOTO 10
      DO 50 J = 1,NJ
      WRITE (6,160) J
      DO 50 KK = 1,NK
      K = NK-KK+1
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      WRITE (6,140) (W(I,J),I=1,NI)
50    CONTINUE
      DO 60 I = 1,NI
      WRITE (6,170) I
      DO 60 KK=1,NK
      K=NK-KK+1
      CALL DAPFROM EM(W,4HRMAT,32*(K-1))
      WRITE (6,140) (W(I,J),J=1,NJ)
60    CONTINUE
      GOTO 10
100   FORMAT(5I0)
110   FORMAT(I0, E0.0, I0)
120   FORMAT( 9H1HEIGHT =,I5,
     122H DAM LEFT FRONT SIDE =,I5,20HDAM LEFT BACK SIDE =,I5,
     212H LEFT BACK =,I5,12HRIGHT BACK =,I5,//)
130   FORMAT(25H NUMBER OF ITERATIONS =,I5,
     19H NUMBOT =,I5,20HMAXIMUM DIFFERENCE =,E13,6)
140   FORMAT(1H ,9E13.6)
150   FORMAT(///11H FOR PLANE ,I3)
170   FORMAT(///11H FOR SIDE  ,I3)
160   FORMAT(///11H FOR FACE  ,I3)
      END
```

```
      SUBROUTINE LAPLACE
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, Z(,)
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK

      CALL INIT MASK
      CALL INIT W
      CALL INIT BOTTOM PLANE
      CALL MAIN LOOP
      DO 1001 K= 2,ZPLANES-2,2
      Z = W(,,K)
      W(MASKRB,K) = W(,,K+1)
      W(MASKRB,K+1) = Z
1001  CONTINUE
      RETURN
      END

C     INITIALIZE THE MASKS
C
      SUBROUTINE INIT MASK
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      LOGICAL T(,)
C
C     MASKMASK IS TRUE IN THE AREA OF W(,,K) WHERE COMPUTATION TAKES PLACE
C

      MASKMASK = ROWS(2,YPOINTS-1) .AND. COLS(2,XPOINTS-1)
C
      T = ROWS(YRPOINTS, YPOINTS) .AND. COLS(XFPOINTS, XPOINTS)
      MASKMASK(T) = .FALSE.
C
C
C     MASKRB IS THE RED/BLACK SCHEME
C
      MASKRB = ALTC(1) .EQ. ALTR(1)
      IF (.NOT.SWITCH(2)) MASKRB=.NOT.MASKRB
      TRACE 127 (MERGE(1,2,TRAN(MASKMASK)))
      TRACE 127 (MERGE(1,2,TRAN(MASKRB)))
      RETURN
      END
```

```
C     INITIALIZE THE W MATRIX
C
      SUBROUTINE INIT W
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, Z(,)
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      REAL TEMPS, TEMPS1

      TEMPS = DAMHEIGHT * DAMHEIGHT * 0.5
      DO 10 K = 1, ZPLANES
      W(,,K) = 0.0
      TEMPS1 = 1 - (K - 1) / EFLOAT(DAMHEIGHT)
      W(,1,K) = TEMPS * TEMPS1 * TEMPS1
10    CONTINUE
      DO 1001 K= 2,ZPLANES-2,2
      Z = W(,,K)
      W(MASKRB,K) = W(,,K+1)
      W(MASKRB,K+1) = Z
1001  CONTINUE
      W(,,ZPLANES+1) = W(,,ZPLANES)
      RETURN
      END
C
C     INITIALIZE THE BOTTOM Z PLANE (I.E. W(,,1))
C
      SUBROUTINE INIT BOTTOM MATRIX
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      LOGICAL MASKRB, MASKMASK
      REAL Z(,), SAVEW(,), ALPHA, BETA
      LOGICAL DONE
      INTEGER NUMBTIMES

      NUMBOT = 0
      ALPHA = OMEGA * 0.25
```

```
      BETA = 1.0 - OMEGA
40    SAVEW = W(,,1)
      NUMBOT = NUMBOT + 1
      DO 45 NUMBTIMES = 1,2
      W(1,,1) = W(3,,1)
      W(YPOINTS,,1) = W(YPOINTS-2,,1)
      Z = W(,,1) + W(=,=,1)
      Z = Z(+,) + Z(,+)
      W(MASKRB.AND.MASKMASK,1) = ALPHA * Z + BETA * W(,,1)
      MASKRB = .NOT. MASKRB
45    CONTINUE
      MAXDIFF = MAX(ABS(SAVEW - W(,,1)))
      DONE=(MAXDIFF.LE.BOTTOMEPSILON).OR.(NUMBOT.GE.BOTTOMMAXITERS)
      IF (.NOT. DONE) GOTO 40
      TRACE 127(MAXDIFF, NUMBOT)
      RETURN
      END
C
C     THE MAIN LOOP - PROCESS ALL THE Z PLANES
C
      SUBROUTINE MAIN LOOP
      COMMON /ISCA/ ZPLANES, XPOINTS, YPOINTS, XFPOINTS, XBPOINTS
      COMMON /ISCA/ YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      COMMON /ISCA/ DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      COMMON /ISCA/ DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, NUMBOT
      INTEGER ZPLANES, XPOINTS, YPTSM2, YPOINTS, XFPOINTS, XBPOINTS
      INTEGER YRPOINTS, YLPOINTS, DAMHEIGHT, DAMFACE, DAMRSIDE
      INTEGER DAMLBACK, DAMRBACK, DAMLFSIDE, DAMLBSIDE
      INTEGER DAMMAXITERS, BOTTOMMAXITERS, NUMBITERS, TIMES
      REAL DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF
      COMMON /RSCA/ DAMEPSILON, BOTTOMEPSILON, OMEGA, MAXDIFF, TIMES
      COMMON /RMAT/W(,,25)
      COMMON /SUBLMAT/MASKRB(,), MASKMASK(,)
      COMMON /WORK/ Z,WK
      LOGICAL MASKRB, MASKMASK
      REAL SAVEW(,), Z(,), Z1(,), WKP1(,), WK(,), MAXD(,)
      REAL MAXSOFAR, ALPHA, BETA, WIDGRID2, WIDTHGRID
      INTEGER NUMBTIMES, TOPPLANE
      LOGICAL TEMPMASK(,), DONE, WSIGN(,), TEST, NOTTEST
      EQUIVALENCE (WSIGN,Z), (Z,Z1), (WK,WKP1)

      ALPHA = OMEGA * 1.0 / 6.0
      BETA = 1.0 - OMEGA
C
C     WIDTH OF GRID (I.E. ONE UNIT SQUARE) IS SET TO 1.0
C
      NUMBITERS = 0
      WIDTHGRID = 1.0
      WIDGRID2 = WIDTHGRID * WIDTHGRID
      TOPPLANE = ZPLANES - 1
```

```
C
C
1       TEST = TIMES .EQ. 1
        NOTTEST = .NOT. TEST
        ITIMES = TIMES
        MAXD = 0.0
C
C
C   ALTER ALL THE ODD NUBERED PLANES!
2       KM2 = 1
        DO 20 K = 2,TOPPLANE,2
        SAVEW = W(:,:,K+1)
        WK = W(:,:,K)
        WK(1,:) = SHN(WK,2)
        WK(YPOINTS,:) = SHS(WK,2)
        Z = WK + WK(:-1,:)
        Z = (Z(:+1,:)+Z(:,:+1)+WK+MERGE(W(:,:,KM2),W(:,:,K+2),MASKRB)-WIDGRID2)*ALPH
        Z(WSIGN) = 0.0
        W(MASKMASK,K+1) = Z
        IF (NOTTEST) GOTO 20
        Z1 = ABS(SAVEW-Z)
        MAXD(Z1.GT.MAXD) = Z1
20      KM2 = K
C
C
C   ALTER ALL THE EVEN NUMBERED PLANES!
        DO 21 K = 2,TOPPLANE,2
        SAVEW = W(:,:,K)
        WKP1 = W(:,:,K+1)
        WKP1(1,:) = SHN(WKP1,2)
        WKP1(YPOINTS,:) = SHS(WKP1,2)
        Z = WKP1 + WKP1(:-1,:)
        Z = (Z(:+1,:)+Z(:,:+1)+WKP1+MERGE(W(:,:,K+3),W(:,:,K-1),MASKRB)-WIDGRID2)*AL
        Z(WSIGN) = 0.0
        W(MASKMASK,K) = Z
        IF (NOTTEST) GOTO 21
        Z1 = ABS(SAVEW-Z)
        MAXD(Z1.GT.MAXD) = Z1
21      CONTINUE
C
C
        ITIMES = ITIMES-1
        IF (ITIMES.GT.1) GOTO 2
        IF (ITIMES.EQ.0) GOTO 3
        TEST = .TRUE.
        NOTTEST = .FALSE.
        GOTO 2
C
3       NUMBITERS = NUMBITERS + TIMES
        MAXDIFF = MAX(MAXD,MASKMASK)
        IF (MAXDIFF.LT.DAMEPSILON) GOTO 4
C
        IF (NUMBITERS.LT.DAMMAXITERS) GOTO 1
4       RETURN
        END
```

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 2170	2. GOVT ACCESSION NO.: AD-A099358
3. RECIPIENT'S CATALOG NUMBER:	
4. TITLE *(and Subtitle)*: THE SOLUTION OF LINEAR COMPLEMENTARITY PROBLEMS ON AN ARRAY PROCESSOR	5. TYPE OF REPORT & PERIOD COVERED: Summary Report - no specific reporting period
	6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): C. W. Cryer, P. M. Flanders, D. J. Hunt, S. F. Reddaway, and J. Stansbury	8. CONTRACT OR GRANT NUMBER(s): DAAG29-80-C-0041 MCS77-26732
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Mathematics Research Center, University of Wisconsin 610 Walnut Street Madison, Wisconsin 53706	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: Work Unit Number 3 - Numerical Analysis & Computer Science
11. CONTROLLING OFFICE NAME AND ADDRESS: See Item 18 below.	12. REPORT DATE: January 1981
	13. NUMBER OF PAGES: 53
14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)*	15. SECURITY CLASS. *(of this report)*: UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

U. S. Army Research Office
P. O. Box 12211
Research Triangle Park
North Carolina 27709

National Science Foundation
Washington, D. C. 20550

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Array processor
Parallel computation
Free boundary problem
Variational inequality
Porous flow
Linear complementarity problem
Numerical solution
Successive over-relaxation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

The Distributed Array Processor (DAP) manufactured by International Computers Limited is an array of 1-bit 200-nanosecond processors. The Pilot DAP on which the present work was done is a 32 × 32 array; the commercially available machine is a 64 × 64 array. We show how the projected SOR algorithm for the linear complementarity problem $Aw \geq b$, $w \geq 0$, $w^T(Aw - b) = 0$, can be adapted for use on the DAP when A is the 'finite-difference' matrix corresponding to the difference approximation to the Laplace operator. Application is made to two linear complementarity problems arising, respectively, from two- and three-dimensional porous flow free boundary problems.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*